MIRO: A DEBUGGING TOOL FOR CLIPS INCORPORATING HISTORICAL RETE NETWORKS

Sharon M. Tuttle
Management Information Systems, University of St. Thomas
3800 Montrose Blvd., Houston, TX, 77006

Christoph F. Eick
Department of Computer Science, University of Houston
Houston, TX 77204-3475

ABSTRACT

At the last CLIPS conference, we discussed our ideas for adding a temporal dimension to the Rete network used to implement CLIPS. The resulting historical Rete network could then be used to store 'historical' information about a run of a CLIPS program, to aid in debugging. MIRO, a debugging tool for CLIPS built on top of CLIPS, incorporates such a historical Rete network and uses it to support its prototype question-answering capability. By enabling CLIPS users to directly ask debugging-related questions about the history of a program run, we hope to reduce the amount of single-stepping and program tracing required to debug a CLIPS program. In this paper, we briefly describe MIRO's architecture and implementation, and the current question-types that MIRO supports. These question-types are further illustrated using an example, and the benefits of the debugging tool are discussed. We also present empirical results that measure the run-time and partial storage overhead of MIRO, and discuss how MIRO may also be used to study various efficiency aspects of CLIPS programs.

1. INTRODUCTION

In debugging programs written in a forward-chaining, data-driven language such as CLIPS, programmers often have need for certain *historical* details from a program run: for example, when a particular rule fired, or when a particular fact was in working memory. In a paper presented at the last CLIPS conference [4], we proposed modifying the Rete network, used for determining which rules are eligible to fire at a given time, within CLIPS, to retain such historical information. The information thus saved using this *historical Rete network* would be used to support a debugging-oriented question-answering system.

Since the presentation of that paper, we have implemented historical Rete and a prototype question-answering system within MIRO, a debugging tool for CLIPS built on top of CLIPS. MIRO's question-answering system can make it much less tedious to obtain historical details of a CLIPS program run as compared to such current practices as rerunning the program one step at a time, or studying traces of the program. In addition, it turns out that MIRO may also make it easier to analyze certain efficiency-related aspects of CLIPS program runs: for example, one can much more easily determine the number of matches that occurred for a particular left-hand-side condition in a rule, or even the number of matches for a subset of left-hand-side conditions (those involved in *beta memories* within the Rete network).

The rest of the paper is organized as follows. Section two briefly describes MIRO's architecture and implementation. Section three then gives the currently-supported question-types, and illustrates how some of those question-types can be used to help with debugging. Empirical results regarding MIRO's run-time and partial storage overhead costs are given in section four, and section five discusses some ideas for how MIRO might be used to study various efficiency aspects of CLIPS programs. Finally, section six concludes the paper.

2. MIRO'S ARCHITECTURE AND IMPLEMENTATION

To improve CLIPS' debugging environment, MIRO adds to CLIPS a question-answering system able to answer questions about the current CLIPS program run. We used CLIPS 5.0 as MIRO's basis. Figure 1 depicts the architecture of MIRO. Because questions useful for debugging will often refer to

historical details of a program run, MIRO extends the CLIPS 5.0 inference engine to maintain historical information about the facts and instantiations stored in the working memory, and about the changes to the agenda. Moreover, in order to answer question-types we provided query-operators that facilitate answering questions concerning past facts and rule-instantiations, and an agenda reconstruction algorithm that reconstructs conflict-resolution information from a particular point of time.

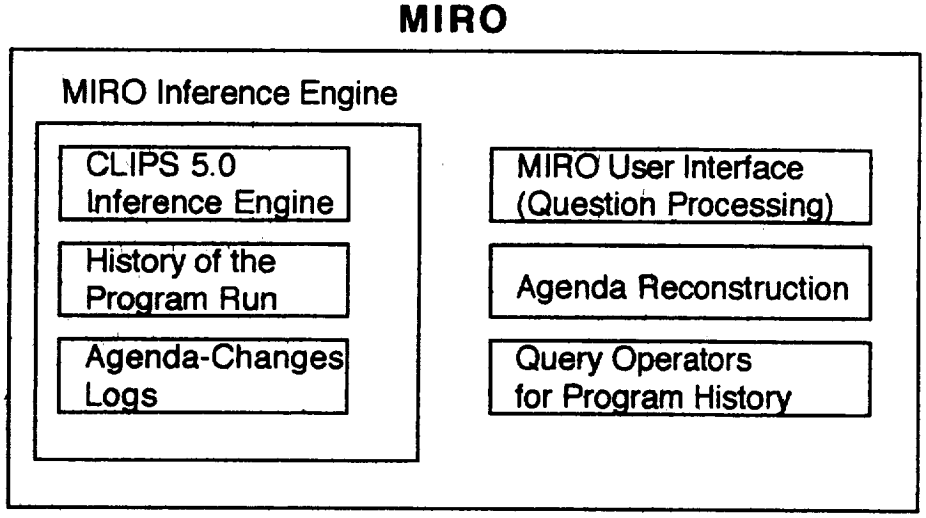

Figure 1 - The MIRO Debugging Environment

One could describe MIRO as a tool for helping programmers to analyze a program run; it assists them by making the acquisition of needed low-level details as simple as asking a question. Where, before, they gathered clues that might suggest to them a fault's immediate cause by searching traces and single-stepping through a program run, now they can simply ask questions to gather such clues. The programmers still direct the debugging process, but the question-answering system helps them to determine the next step in that process. By allowing programmers to spend less time single-stepping through program runs and searching traces for historical details, this question-answering system can let programmers save their attention and energy for the high-level aspects and intuition needed in debugging.

As already mentioned, CLIPS 5.0 forms the basis of MIRO: the CLIPS 5.0 source code was modified and augmented to implement MIRO. To quickly obtain an operational prototype, we used existing code whenever possible, and we patterned the historical Rete and agenda reconstruction additions after those used for regular Rete within CLIPS. This software reuse included replicating the code for existing data structures when creating new data structures, augmenting existing data structures, calling existing functions from new functions whenever possible, and modifying existing functions when their functionality was almost, but not quite, what was needed.

So, when implementing, for example, historical Rete's past partitions, the past partitions were patterned after the current (formerly only) partitions. The data structures for facts, instantiations, and rule instantiations were all augmented with time-tags, and rule instantiations were also augmented with a fired flag, set if that rule instantiation was actually fired. We added analogous functions to those used for maintaining the current working memory for maintaining the past working memory, and so on. This approach reduced programming overhead, and, because CLIPS 5.0 was already quite efficient, the added historical components were also quite efficient.

We also added code to measure various run-time characteristics, such as the number of current and past facts, rule instantiations, and Rete memory instantiations, to better compare program runs under regular CLIPS and MIRO, as shown at the end of section four.

The bulk of MIRO was implemented over the course of a single year, by a single programmer; however, other research-related activities were being done concurrently with this development. It

probably took about seven programmer-months to bring MIRO to the point where it had 19 question-types. Note, however, that this does not take into account the time spent designing the historical Rete and agenda reconstruction algorithms.

The version of CLIPS 5.0 that we developed MIRO from, and that we used for comparisons with MIRO, contained 80497 lines of code and comments; this includes the source code files, include files, and the UNIX make file. The same files for MIRO contained 86099 lines, also including comments; so, we added about 5600 additional lines of code and comments, making MIRO about 7% larger than CLIPS 5.0.

3. MIRO'S QUESTION TYPES

Adding the question-answering system itself to MIRO was as easy as adding a new user-defined function to the top-level of CLIPS; we constructed a CLIPS top-level function called askquestion. The difficult part was determining the form that this question-answering would take. Our primary goals of constructing a prototype both to demonstrate the feasibility of, and to illustrate how, the information from the historical Rete network and other history-related structures could be used to answer programmers' debugging-related questions had a strong impact on the design that we decided to use.

We assume that the kinds of questions that can be asked are limited to instances of a set of fixed question-types; each question-type can be thought of as a template for a particular kind of question. This allows the question-answering system to have a modular design: each question-type has an algorithm for answering an instance of that type. This also allows additional question-types to be easily added, and to be tested as they are added, as it is discovered which are desirable for debugging.

The design of the interface for getting programmers' questions to the explanation system is a worthy research topic all by itself; to allow us to concentrate more on answering the questions, we use a very simple interface, with the understanding that future work could include replacing this simple interface with a more sophisticated front-end. Since we are more interested in designing a tool for debugging and less interested in natural language processing, the question-answering system uses a template-like approach for the various question-types that the programmers will be able to ask. That is, each question-type has a specific format, with specific ``blanks" which, filled in, result in an instance of that question-type. Furthermore, to avoid requiring the programmers to memorize these question-type formats, we use a menu-based approach: when the programmers enter the command (askquestion), a list of currently-supported question-types is printed on-screen. They then enter the number of the desired question-type, and the explanation system queries them to fill in that question-type's necessary blanks.

Implementing this approach was quite straightforward, because regular CLIPS already has some tools for obtaining top-level command arguments. We only had to modify them a little to allow for the optional askquestion arguments. A main askquestion function prints the menu if no arguments are given, and then asks which question-type is desired; a case statement then uses either that result or the first askquestion argument to call the appropriate question-answering function, which is in charge of seeing if arguments for its needed values have already been given, or must be asked for. After each particular question-type's question-answering function obtains or verifies its needed values, it then tries to answer the question, and print the result for the programmer.

We currently support 19 question-types, as shown in Figure 2. However, question 9, why a particular rule did not fire, currently only tells if that rule was eligible or not at the specified time.

1. What fact corresponds to fact-id <num>?
2. When did rule <name> fire?
3. What rule fired at time <num>?
4. What facts were in memory at time <num>?
5. How many current facts are in memory now?
6. How many past facts are in memory now?
7. How many current rule activations are on the agenda now?
8. How many past rule activations are in memory now?

```
9.  Why did rule <name> not fire at time <num>?
10. How many current alpha instantiations are in memory now?
11. How many past alpha instantiations are in memory now?
12. How many current beta instantiations are in memory now?
13. How many past beta instantiations are in memory now?
14. What are the Rete memory maximums?
15. What were the agenda changes from time <num> to time <num>?
16. How many current not-node instantiations are in memory now?
17. How many past not-node instantiations are in memory now?
18. What was the agenda at the end of time <num>?
19. How many agenda changes were there from time <num> to time <num>?
```

Figure 2 - Currently-Supported Question-Types in MIRO

We will now give some examples of MIRO's question-answering system at work. We will describe some hypothetical scenarios, to illustrate how MIRO might be useful in debugging; the responses shown are those that would be given by MIRO in such situations.

Consider a program run in which the program ends prematurely, that is, without printing any output. One can find out the current time-counter value with a command that we added to MIRO specifically for this purpose --- if one types (time-counter) after 537 rule-firings, it prints out:

```
time_counter is: 537
```

The programmers can now ask, if desired, which rules fired at times 537, 536, 535, etc. If they type (askquestion), the menu of rules will be printed; if they choose question-type number 3, "What rule fired at time <num>?", then it will ask what time-counter value they are interested in; if 537 is entered, and if a rule named "tryit" happened to be the one that fired at that time, then MIRO would print an answer like:

```
Rule tryit fired at time 537
```

with the following rule activation:

```
0     tryit: f-30,f-15,f-47 time-tag: (530 *)(activn time-tag: 530 537))
```

This tells the programmers that rule tryit fired at time 537, and that the particular instantiation of rule tryit that fired had LHS conditions matched by the facts with fact-identifiers f-30, f-15, and f-47. This instantiation of rule tryit has been eligible to fire since time 530 --- before the 531st rule firing --- but, as shown, the rule instantiation's, or activation's, time-tag is now closed, with the time 537, because it was fired then, and a rule that is fired is not allowed to fire again with the same fact-identifiers.

Now, if the programmers suspect that this rule-firing did not perform some action that it should have performed --- to allow another rule to become eligible, for example --- then they can use the regular CLIPS command "pprule" to print the rule, so that they can examine its RHS actions. If it should not have fired at all, then they may wish to see why it was eligible. For example, in this case, they may want to know what facts correspond to the fact-identifiers f-30, f-15, and f-47. One can look at the entire list of fact-identifiers and corresponding facts using the regular CLIPS (facts) command, but if the program has very many facts, it can be quite inconvenient to scroll through all of them. So, MIRO provides the question-type "What fact corresponds to fact-id <num>?". On first glance, this question appears to have no historical aspect at all; however, it does include the time-tag for the instance of the fact corresponding to this fact-identifier. This can be helpful to the programmers, if they suspect that one of the facts should not have been in working memory --- then, the opening time of that fact's time-tag can be used to see what rule fired at that time, probably resulting in this fact's addition. Since this question-type is the first in the list, and requires as its only additional information the number of the fact identifier whose fact is desired, typing (askquestion 1 47) will ask the first question for fact-identifier f-47, giving an answer such as:

```
Fact-id 47 is fact:
   (p X Y) time-tag: (530 *)
```

If the programmers suspect that fact f-47, now known to be (p X Y), should not be in working

memory --- if they think that it is a fault that it exists, and is helping rule tryit to be able to fire --- then they can again ask the question-type "What rule fired at time <num>?" to see what rule fired at time 530, when this instance of (p X Y) joined working memory. They can then see if the rule that fired at time 530 holds the cause of the fault of (p X Y) being in working memory, and enabling the faulty firing of tryit at time 537.

4. COMPARISONS BETWEEN MIRO AND CLIPS 5.0

As already mentioned, we implemented MIRO by starting with CLIPS 5.0; we then generalized its Rete inference network [2] into a *historical Rete* network, added an agenda reconstruction capability, and added the prototype question-answering capability. Historical Rete and agenda reconstruction are discussed in more detail in [5] and [6]. We also made some other modifications, to allow for experimental measures; for example, we added code to measure various run-time characteristics such as the number of current and past instantiations, and the number of current and past facts. We then ran a number of programs under both MIRO and CLIPS 5.0.

The programs that we used range fairly widely in size, and behavior. Four of the programs --- dilemma1, mab, wordgame, and zebra --- are from CLIPS 5.0's Examples directory. The dilemma1 program solves the classic problem of getting a farmer, fox, goat, and cabbage across a stream, where various constraints must be met. The mab program solves a planning problem in which a monkey's goal is to eat bananas. The wordgame program solves a puzzle in which two six-letter names are "added" to get another six-letter name, and the program determines which digits correspond to the names' letters. Finally, the zebra program solves one of those puzzles in which five houses, of five different colors, with five different pets, etc., are to each have their specific attributes determined, given a set of known information.

The AHOC program was written by graduate students in the University of Houston Department of Computer Science's graduate level knowledge engineering course COSC 6366, taught by Dr. Eick in Spring 1992. AHOC is a card game with the slightly-different objective that the players seek to win *exactly* the number of tricks bid. The program weaver ([1], [3]) is a VLSI channel and box router; we obtained a CLIPS version of this program from the University of Texas' OPS5 benchmark suite. Finally, can_ordering_1 is a small program that runs a rather long canned beverage warehouse ordering simulation, also from the Spring 1992 COSC 6366 knowledge engineering class; it was written by C. K. Mao.

We ran each program three times under MIRO and under CLIPS 5.0, on a Sun 3 running UNIX, with either no one else logged in, or one other user who was apparently idle. For every run in both CLIPS and MIRO, we used the (watch statistics) command to check that the same number of rules fired for each run of the program; for every MIRO run of a program, we also made sure that all runs had the same number of instantiations at the end of the run. The run-times are given in Table I.

The run-times for programs run using MIRO were usually only slightly slower than those using regular CLIPS 5.0; one program, mab, took 11.4% more time in MIRO, but on average, the MIRO runs only took 4.1% more time. Interestingly enough, AHOC ran, on average, slightly faster under MIRO than under regular CLIPS. This could be because regular CLIPS 5.0 returns the memory used for facts and instantiations to available memory as they are removed, which MIRO does not do until a reset or clear, because such facts and instantiations are instead kept, and moved into the past fact list or past partitions. The average 4.1% additional time required by MIRO to run a program seems quite reasonable, especially since the additional time is only required while debugging, and allows programmers the benefits of the MIRO tool.

Another feature of the (watch statistics) command under regular CLIPS, besides printing the number of rules fired and the time elapsed, is that it also prints the average and maximum number of facts and rule instantiations during those rule-firings. We enhanced this command in MIRO so that it also keeps track of the average and maximum number of past facts and past rule instantiations, as well as the averages and maximums for different kinds of Rete memory instantiations, both past and current. To get some idea of the overhead needed to store historical information from a run, we compared the average number of current facts during a run to the average number of past facts, and compared the

Table I
Run-Times

program	which CLIPS	# rules fired	run times (in sec.) run1	run2	run3	avg run-time	% slower
dilemma1	regular	80	0.80	0.88	0.80	0.83	
	MIRO	80	0.88	0.84	0.94	0.89	7.3%
mab	regular	81	2.02	1.94	2.06	2.01	
	MIRO	81	2.26	2.18	2.28	2.24	11.4%
wordgame	regular	102	16.84	16.50	16.58	16.64	
	MIRO	102	16.78	16.74	16.62	16.71	0.4%
zebra	regular	28	7.70	7.02	6.96	7.23	
	MIRO	28	7.88	7.16	7.20	7.41	2.5%
AHOC	regular	2747	60.34	60.28	60.52	60.38	
	MIRO	2747	59.10	58.62	60.28	59.33	-1.8%
weaver	regular	745	198.26	198.94	201.24	199.48	
	MIRO	745	209.10	208.84	207.94	208.63	4.6%
can_ordering_1	regular	3392	174.20	174.26	179.20	175.89	
	MIRO	3392	186.18	181.34	181.58	183.03	2.1%

average number of current rule instantiations to the average number of past rule instantiations.

Another feature of the (watch statistics) command under regular CLIPS, besides printing the number of rules fired and the time elapsed, is that it also prints the average and maximum number of facts and rule instantiations during those rule-firings. We enhanced this command in MIRO so that it also keeps track of the average and maximum number of past facts and past rule instantiations, as well as the averages and maximums for different kinds of Rete memory instantiations, both past and current. To get some idea of the overhead needed to store historical information from a run, we compared the average number of current facts during a run to the average number of past facts, and compared the average number of current rule instantiations to the average number of past rule instantiations. (Analogous comparisons for different kinds of Rete memory instantiations, as well as comparisons of the maximum number of current items to the maximum number of past items, can be found in [6]).)

Table II shows these averages for facts and rule instantiations. It also gives, where appropriate, how many times bigger the average number of past items is than the average number of current items. Compared to the average number of current items during a run, fewer times as many past facts than past rule instantiations will need to be kept. This suggests that there is more overhead to storing rule instantiation history than there is to storing fact history.

Table II shows that the space to store past facts and rule instantiations, compared to the average space to store current facts and rule instantiations, can, indeed, be high, especially for long runs. But, long runs should be expected to have more historical information to record. Also, it should be noted that in running these programs on a Sun 3, we never ran into any problems with space, even with the additional historical information being stored. During program development, the storage costs should be acceptable, since they facilitate debugging-related question-answering. The programmers can also limit the storage costs by only running their program using MIRO when they might want to obtain the answers to debugging-related questions; at other times, they can run their program using regular CLIPS.

5. USING MIRO IN STUDYING CLIPS PROGRAMS

We hope that MIRO's question-answering system can make debugging a CLIPS program easier and less tedious. Here, we consider another use of MIRO: to analyze certain performance aspects of CLIPS programs. With a shift of viewpoint, such analysis may be involved in a variant of debugging -

Table II
Average No. of Facts and Rule Instantiations

program	avg. # of facts		times	avg. # rule insts		times
	current	past	bigger	current	past	bigger
dilemma1	11	20	1.82	6	72	12.00
mab	19	42	2.21	1	64	64.00
wordgame	71	0		49	51	1.04
zebra	78	14	0.18	11	14	1.27
AHOC	231	792	3.43	107	3447	32.21
weaver	151	383	2.54	6	689	114.83
can_ordering_1	113	1639	14.50	10	2795	279.50

-- if, for example, a program takes so long to run that the programmers consider the run-time to be a problem, then such performance analysis may help in determining how to modify the program so that it takes less time. Such aspects may also be of interest in and of themselves, both to programmers and to researchers studying the performance aspects of forward-chaining programs in general.

Note that a Rete network, historical or regular, encodes a program's rules' LHS conditions; the RHS conditions of those rules are not represented, except perhaps by pointers from a rule's final beta memory to its actions, to help the forward-chaining system to more conveniently execute the RHS actions of a fired rule. So, MIRO could be helpful primarily in analyzing the efficiency involved in the match step of the recognize-act cycle. Being able to locate Rete memories whose updating could cause performance trouble-spots could be useful for improving the overall performance of a CLIPS program.

Using MIRO, it is much easier to discover some of the dynamic features of a CLIPS program run, such as the number of instantiations within the network during a run. One can study worst-case and average-case behavior within a CLIPS program run by looking at the number of average, and maximum, facts, rule instantiations, and alpha, beta, and not-memory instantiations. For example, a great disparity between the average number of current beta instantiations, and the maximum number reached during a run could indicate volatility in beta memory contents that could have a noticeable performance impact.

One might consider the total number of changes to a beta memory during a run to be the total number of additions to and deletions from that memory --- or, the total number of current instantiations at the end plus two times the number of past instantiations. Averaged over the number of rule-firings, this would give the number of beta memory changes per rule-firing, which, if high, might very well correlate with more time needed per rule-firing; and, averaged over the number of total beta memories, this would give us a rough average of the number of changes per beta memory. We could even determine a number of working memory changes this way, by adding the number of current facts to two times the number of past facts, and use this to obtain an average number of fact changes per rule firing.

Another measure that might be very telling would be the average number of memory changes per rule-firing, computed by counting each past instantiation at the end of a run as two memory changes --- since each was first added to a current instantiation, and then moved to a past partition --- and each current instantiation at the end of a run as a single memory change, and then dividing the sum by the number of rule-firings. We can also determine the average number of fact changes per rule-firing similarly. In measurements we made using the seven programs discussed in the previous section, the most important factor that we found that correlated with performance was that a high number of instantiation changes per rule-firing did seem to correlate with more time needed per rule-firing [6].

Although one can reasonably obtain some of the information mentioned above by using regular CLIPS, much of it would be very inconvenient to gather using it. For example, determining the

average and maximum number of past facts and past rule instantiations would be difficult to obtain using regular Rete. We modified the existing CLIPS (watch statistics) command in MIRO so that it also keeps track of this additional information.

Note that historical Rete can also be used to support trouble-shooting tools and question-types, in addition to supporting question-answering for debugging. For example, any time that the programmer can specify a particular time of interest, historical Rete searches can be made that focus only on instantiations in effect at that time.

The discussed examples show the potential that MIRO has as a tool in analyzing CLIPS program performance, as well as in debugging. Information about what occurred during Rete network memories can be more reasonably retrieved, making such analysis more practical, across larger samples of programs. Note, too, that a programmer can choose to look at averages over all of a program's memories, or for a single memory, or for a particular rule's memories, as desired.

6. CONCLUSIONS

In this paper, we have followed up on our work reported at the last CLIPS conference, describing how we have implemented historical Rete and question-answering for debugging in MIRO, a debugging tool built on top of CLIPS 5.0. We have described MIRO, and have hopefully given a flavor for how it may be used to make debugging a CLIPS program easier and less tedious, by allowing programmers to simply ask questions to determine when program events --- such as rule firings, or fact additions and deletions --- occurred, instead of having to depend on program traces or single-stepping program runs. We have further described how MIRO might be used to study certain performance-related aspects of CLIPS programs.

The empirical measures included also show that MIRO's costs are not unreasonable. Comparisons of programs run in both MIRO and CLIPS 5.0, which MIRO was built from, have been given; on average, the run-time for a program under MIRO was only 4.1% slower than a program run under CLIPS 5.0, when both were run on a Sun 3. Comparing the average number of past facts and rule instantiations to the average number of current facts and current rule instantiations, there were, on average, 3.53 times more past facts than current facts, and 72.12 times more past rule instantiations than current rule instantiations. But this historical information permits the answering of debugging-related questions about what occurred, and when, during an expert system run. We also gave examples of the kinds of question-types that MIRO can currently answer, as well as examples of the kinds of answers that it gives. We hope that this research will encourage others to also look into how question-answering systems can be designed to serve as tools in the development of CLIPS programs.

REFERENCES

1. Brant, D. A., Grose, T., Lofaso, B., Miranker, D. P., "Effects of Database Size on Rule System Performance: Five Case Studies," Proceedings of the 17th International Conference on Very Large Data Bases (VLDB), 1991.

2. Forgy, C. L., "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," ARTIFICIAL INTELLIGENCE, Vol. 19, September, 1982, pp. 17-37.

3. Joobbani, R., Siewiorek, D. P., "WEAVER: A Knowledge-Based Routing Expert," IEEE DESIGN AND TEST OF COMPUTERS, February, 1986, pp. 12-23.

4. Tuttle, S. M., Eick, C. F., "Adding Run History to CLIPS," 2nd CLIPS Conference Proceedings, Vol. 2, Houston, Texas, September 23-25, 1991, pp. 237-252.

5. Tuttle, S. M., Eick, C. F., "Historical Rete Networks to Support Forward-Chaining Rule-Based Program Debugging," INTERNATIONAL JOURNAL ON ARTIFICIAL INTELLIGENCE TOOLS, Vol. 2, No. 1, January, 1993, pp. 47-70.

6. Tuttle, S. M., "Use of Historical Inference Networks in Debugging Forward-Chaining Rule-Based Systems," Ph.D. Dissertation, Department of Computer Science, University of Houston, Houston, Texas, December, 1992.